Developing a Product Line Approach for Flight Software

Mike Stark, Dave McComas

NASA/Goddard Space Flight Center

Guilherme H. Travassos

COPPE/Federal University of Rio de Janeiro

Maurizio Morisio

Politecnico di Torino

Abstract

This paper describes research into software product lines performed jointly at Goddard Space Flight Center by the Flight Software Branch and the Software Engineering Laboratory. It describes the motivation for this project, the approach taken to the research, and some observations from our initial prototyping.

Introduction

This paper discusses the development and evaluation of a product line development approach within the Flight Software Branch (FSB) at Goddard Space Flight Center (GSFC). The work described here is based on both previous experiences with software reuse at GSFC and product line methodologies documented in the software engineering literature. The problem domain used for this evaluation is Guidance, Navigation and Control (GNC). The reasons for choosing this problem are discussed in the next section.

The methodologies investigated by this project include Synthesis as documented by the Software Productivity Consortium [1], Family-Oriented Abstraction, Specification and Translation (FAST) [2], Product Line Software Engineering (PuLSE) from the Fraunhofer Center [3], and Feature-Oriented Domain Analysis (FODA) as documented by the Software Engineering Institute [4]. Previous work at GSFC includes investigations within the FSB [5], and the Generalized Support Software (GSS) project [6], which developed reusable GNC software for ground-based applications. GSS largely predated the popularity of "product lines" as a software engineering approach, but shared many of the characteristics of current product line work. Our processes are primarily based on Synthesis and FAST.

One characteristic that can be discerned from reading the product line literature is that there is no common definition for software product line. We will adopt the following definition from Weiss's work [2]: "We call a family of products designed to take advantage of the common aspects and predicted variabilities *a product line.*"

A second characteristic of product line processes is that at the highest level they look extremely similar. The typical product line process has separate activities associated with *domain engineering*, which produces a set of reusable assets, and *application engineering*, which configures these assets into application software products. Finally, there is a mapping called a *decision model* that enables an application engineer to examine the reusable assets and readily specify how they are to be used for a specific

application system. Figure 1 represents the relationship between domain engineering processes and products, application engineering processes and products, and the decision model.

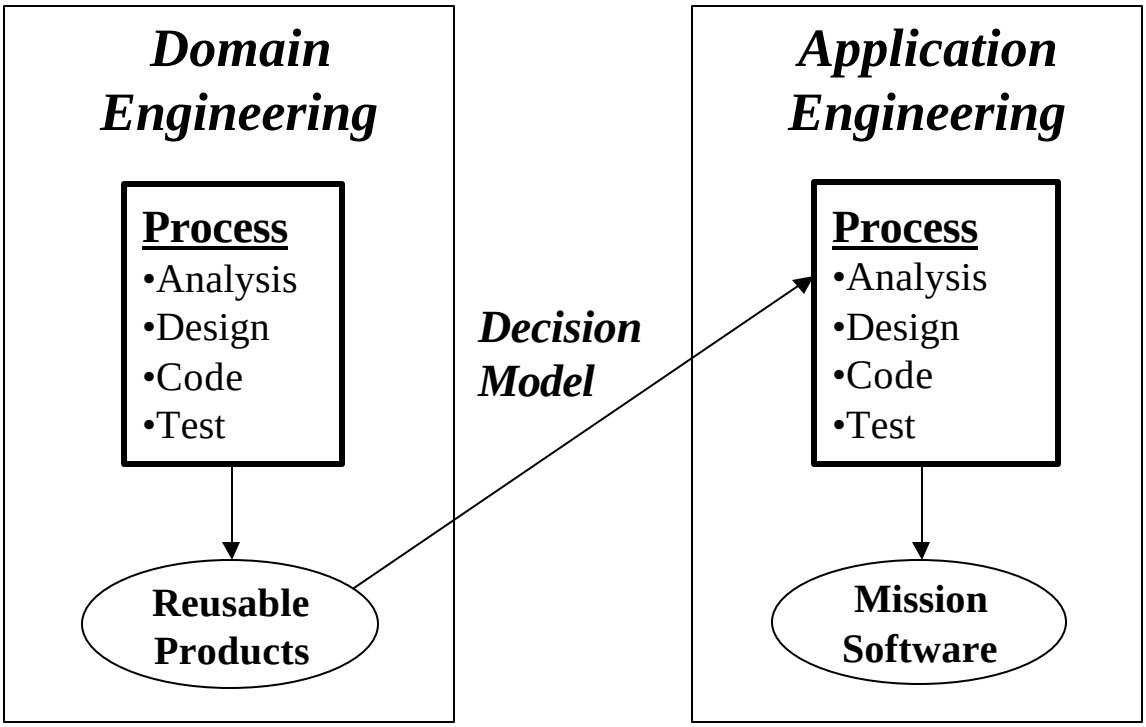

Figure 1. Product Line Development

For the FSB, the following benefits were identified as being attainable using such approach:

- To apply technology consistently across all projects;
- To control product growth;
- To reduce development time without increasing risk, and;
- To institutionalize knowledge that is the legacy of the four predecessor organizations.

The next three sections of this paper describe the FSB's reasons for examining a product line approach, the details of how the project is approaching the problem, and observations about what has happened so far, respectively. The final section summarizes our conclusions and suggests future directions.

Flight Software Branch Background

The Flight Software Branch was formed in December 1997 as part of a major reorganization at GSFC. This branch combined personnel from four different predecessor organizations, each of which had its own distinct culture. The history of flight software projects at GSFC is that commitments to missions are met, but they are often met at a high cost in dollars and overtime. One could characterize these projects as operating successfully at CMM level 1, with a high dependence on skilled "heroes" who

make things work. These factors, coupled with shrinking budgets and schedules, motivate the Flight Software Branch to consider using a product line approach.

To evaluate product line approaches, the FSB decided to use the guidance, navigation and control problem domain. There are several reasons for choosing this domain. First, GNC is a very mature, stable problem domain. It is rooted in the physics of Newtonian mechanics and the mathematics of parameter estimation and control theory. Typically the most variable part of the domain is the modeling of spacecraft sensors and actuators, but even this area works with a finite set of hardware models.

Second, GNC is needed for every satellite mission, and in most cases the FSB has ultimate responsibility for this software. This is distinct from current ground systems, where the software is often acquired rather than developed under NASA control. The FSB has a set of about a dozen recent missions that can be used as input to domain analysis.

These missions include both Earth and space science missions, and have several high-level characteristics that are driven by the scientific requirements. Typical characteristics include the distance of the spacecraft's orbit from the earth and required pointing accuracy. These characteristics drive both the overall design of the mission and the specifics of the GNC flight software, and a product line must be flexible enough to support a wide range of mission goals.

In addition to the FSB experience with GNC flight software, choosing GNC as a problem domain allows lessons to be drawn from the GSS project's experience with GNC ground software. This project was able to use object-oriented technology to realize the development cost and schedule goals set by management. These strengths are part of the motivation for the use of object-oriented approaches described in the next section. Despite these strengths, GSS is not currently being used by new projects. This is at least in part due to the focus on optimizing the process from a developer point of view, without regard to whether the processes and products addressed the needs of stakeholders such as flight dynamics mathematical analysts. The considerations described above led to the following project goals:

- *Provide a framework for continuous improvement*. In the current mode of development for the FSB, it is very difficult to assign the most skilled personnel to research technology improvements, as they are required to meet mission requirements. Even this project had episodes where problems on mission projects diverted key personnel who were in principle assigned full time to look at improving FSB development practices. An effective product line would help free resources for research and development projects, and would provide a baseline for technologists who are evaluating new technologies for potential benefits.
- *Reduce development time without sacrificing quality.*
- *Increase productivity.*

  These two goals allow projects to be supported by smaller teams, or would allow the same size teams to develop more complex software, reusing quality software artifacts already built to capture domain commonalities.

- *Provide rapid prototyping capabilities*. The FSB has been interested in integrating their models with code generated by analysis tools such as MATLAB. This would enable controls analysts to use the same high fidelity models that are used in flight software within their early mission studies. This is not the main goal of the project, but if this is shown to be feasible it would make the product line useful to a larger community of GNC engineers than just the flight software developers.

Together, these goals form an ambitious agenda. The purpose of the product line research is to evaluate whether the FSB can accomplish these goals. The next section of this paper describes the approach to determining whether these goals can be met.

Product Line Development and Evaluation Approach

To succeed in meeting the project goals, the product line research must show that the approach being studied is technically feasible, that it is cost effective, and that it serves the needs of the key stakeholders in a flight software project. The set of stakeholders under consideration include mission flight dynamics and controls analysts, mission test and maintenance personnel, and the domain engineers themselves.

The approach we took towards developing and assessing a product line consisted of three steps. First, we evaluated existing product line methods via review of the literature and using training opportunities that were available. Second, we performed a high-level domain assessment to determine the boundaries of the GNC domain for this product line. These first two steps established a framework for the product line development process and the boundaries of the problem domain.

After these initial steps, the team implemented a series of prototypes to evaluate both the products and processes that comprise the product line. At the writing of this paper, the team was nearing completion of the second prototype application.

While these research activities are presented sequentially, in practice there is much iteration between them. For example, as more process lessons are learned during prototyping, the previously evaluated processes can be re-examined and refined in light of the new knowledge. The next three sections describe each stage of development and evaluation in more detail.

Method Evaluation

There are different product line approaches described in the technical literature, as is discussed in the introduction to this paper. To identify that one that best fits our project needs would imply in the accomplishment of a detailed evaluation process, which we did not have time and enough people to accomplish. Therefore, since we had training resources immediately available for Synthesis, we initially attempted to apply Synthesis. However, we found that the further one went in the Synthesis process, the fewer details were provided by the documentation. Supplementing Synthesis with ideas from FAST helped address this lack of detail. A further issue was that Synthesis introduced a completely new set of terminology, which was difficult to understand in the context of the current flight software development process.

The solution we applied was to create a project-specific process framework, as shown in Figure 2 below. This process merges the traditional approach of doing analysis, design, implementation and test with the concepts of domain and application engineering shown in Figure 1. This framework identifies the high-level processes and products for both domain and application engineering. This approach is described further in [8] and [9].

The details of each process step and product will be elaborated as part of the prototyping process; at this point the domain and application analysis processes have been addressed more thoroughly than the others. The domain analysis approach uses the Unified Modeling Language (UML) notation, with extensions added to represent variabilities [8].

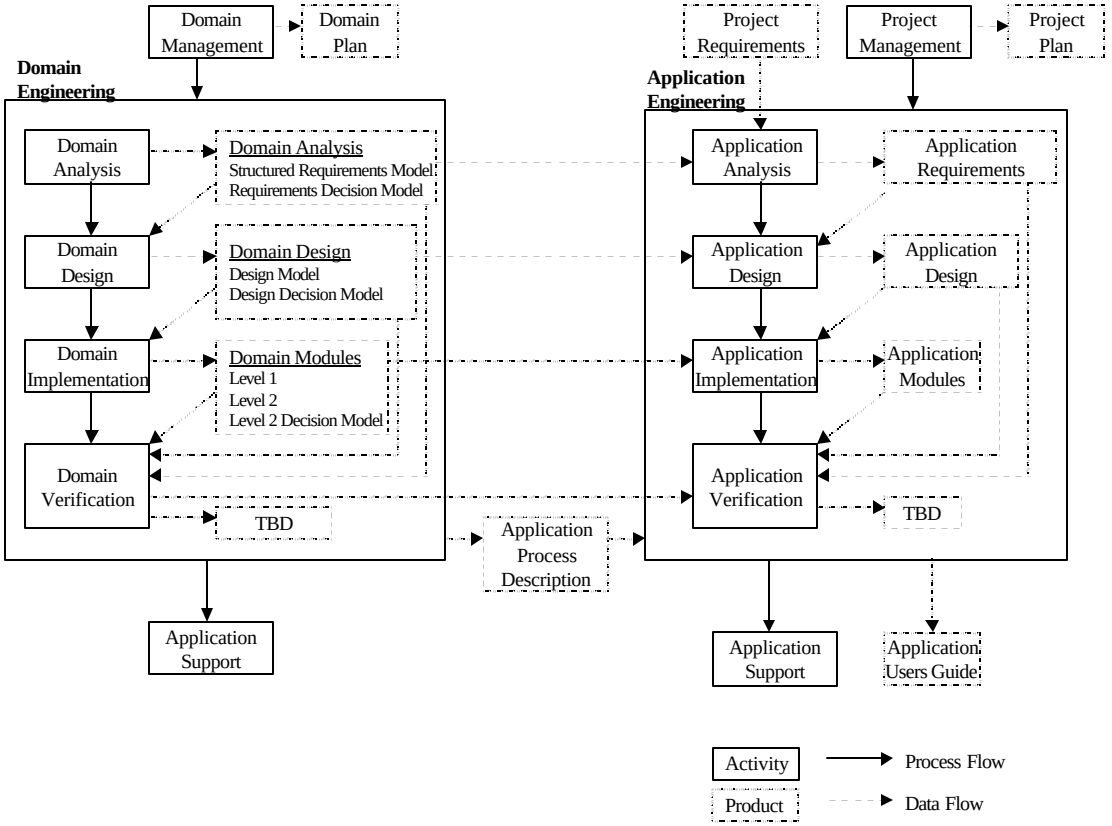

Figure 2. Product Line Process Framework

This process framework can also be compared to the GSS processes. On the domain engineering side, the analysis, design and test were similar (at this highly abstract level), but the domain verification was generally done through design and code inspections, and extensive functional testing of the initial application that used a domain asset. On the application engineering side, GSS merged the application design and implementation into a single configuration process that was a highly streamlined way of creating an application from a standardized mission specification.

The GSS processes were successful in that they allowed applications to be built more rapidly and at a lower cost, so the high-level similarities are encouraging. At the detailed level, however, it will be necessary to create reusable assets that are more usable by analysts and testers than the GSS asset library.

High-level Domain Assessment

The process evaluation discussed above gave a framework for the elaboration of product line process elements. The high-level domain assessment provides a framework for the GNC domain itself. The key elements of this assessment were to

- Bound the domain
- Identify the essential subdomains
- Establish priorities for prototyping

All of these were done using information from a set of reference missions that are under development or maintenance by the FSB. These missions were selected to be representative of the anticipated future missions.

The notation used to represent the domain boundaries and the essential subdomains included a context diagram, an informal subdomain dependency diagram, and documentation of assumptions about commonalities, variabilities and exclusions. Figures 3 and 4 show the context diagram and the dependency diagram, respectively.

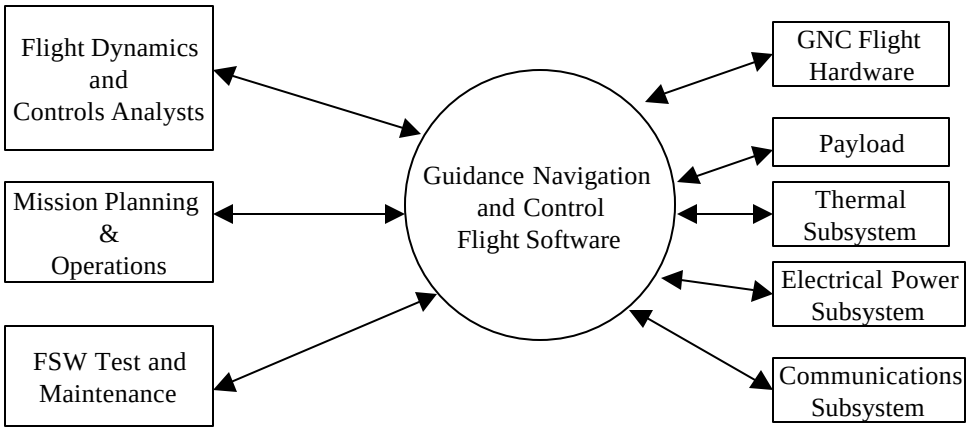

Figure 3. Context diagram for Guidance, Navigation and Control

In Figure 3, the boxes represent external entities (both people and other spacecraft subsystems), with the arrows representing data flows between the GNC flight software and these external entities. In Figure 4 the boxes represent subdomains within the GNC domain, and the arrows represent dependencies. The dotted lines are added to indicate layering within this model.

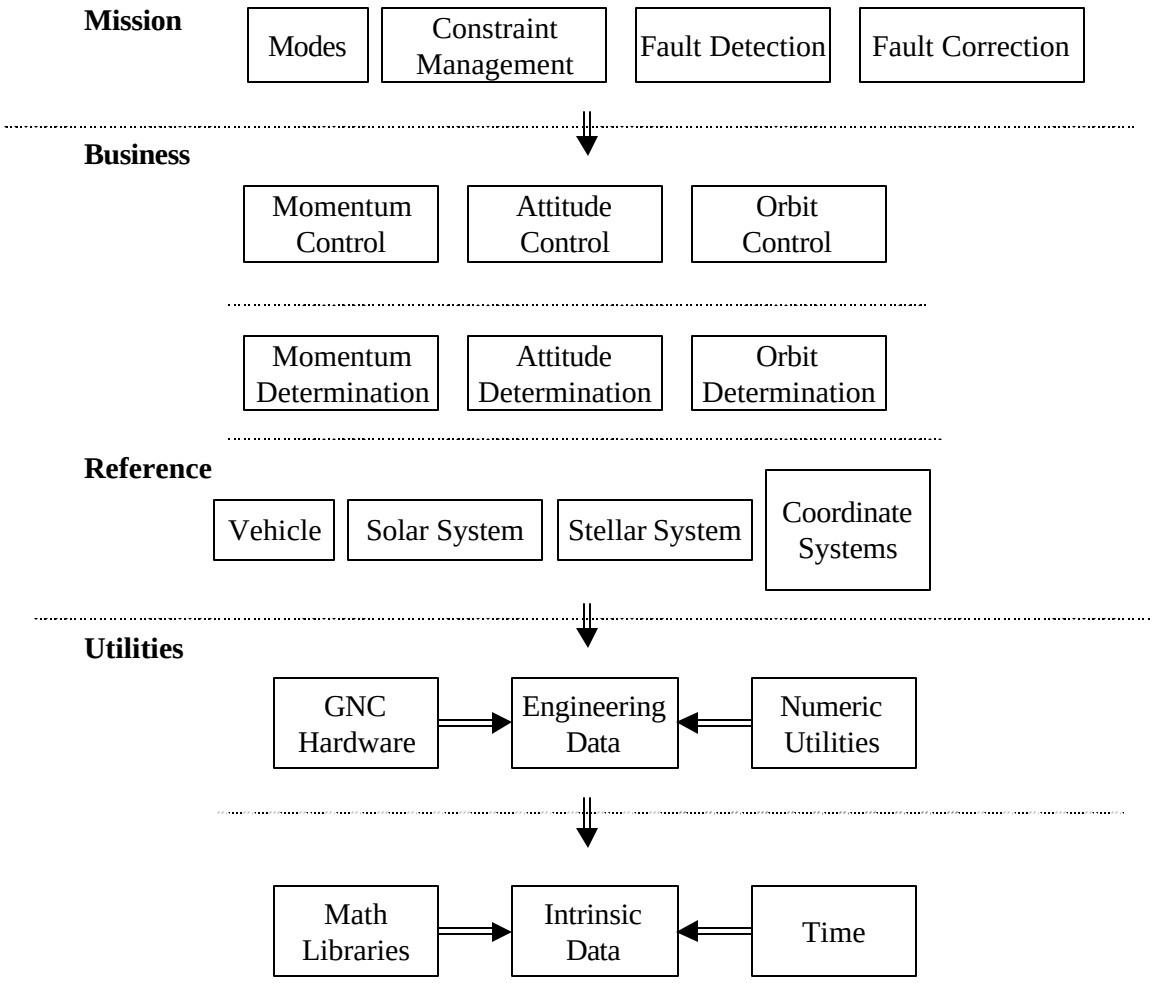

Figure 4. Subdomain dependencies

The commonalities, variabilities and assumptions are documented using the text format template described for Synthesis and FAST. A simple example of the format is shown below.

Commonality/Variability/Exclusion notation

Solar Ephemeris Model Commonality Assumptions

1. Compute solar position in J2000 using the current spacecraft time.

   Justification: Position is the minimal required state information.

Solar Ephemeris Model Variability Assumptions

1. Compute solar velocity.

   Justification: Compute only if mission requires it.

Solar Ephemeris Model Exclusion Assumptions

1. Solar velocity with respect to the stars is not required since star catalogs take this into account.

The way these assumptions are documented is very similar to the text notation already used for mission requirements. Thus the domain engineering team believes that this notation will be relatively easy for FSB developers to adopt.

Prototyping

Once the process framework was defined and the initial domain assessment completed, the team defined a series of prototype applications. Each prototype is defined to analyze the following issues (as shown in Table 1 below):

- Domain functionality
- Flight software architecture
- Product line processes

Functionality	Architecture	Processes
1. *Orbit Propagation* – basic models of spacecraft and celestial body motion	Windows NT executable, open loop system.	Exploration of UML for domain analysis
2. *Safe Hold control* – adds sensor/actuator hardware and simple control algorithm.	Windows NT executable, closed loop system (includes simulated response to control commands), Evaluate approach to integrating hardware-dependent code	Identify design patterns created during prototype development.
3. *Attitude determination* Add sun sensor, magnetometers, sun and magnetic field models, attitude determination algorithms	Windows NT executable, incorporate simulated flight data system (FDS) environment.	Document and apply process for developer perspective
4. *Inertial Hold control* Add inertial hold control algorithm	Add multiprocessing, run on flight hardware testbed if available	Document and apply process for analyst and tester perspectives

Table 1. Prototype planning.

The prototyping so far has evaluated several potential processes and product notations beyond what was specified for Synthesis. These products include UML class diagrams, mission/client matrices, and mission/capability matrices.

Figure 5 is the class diagram for the Celestial Body subdomain that was used for the orbit propagation prototype. The UML notation is extended with a variability stereotype, indicated by the <<V>> annotation. The <<V>> indicates where an application engineer

would choose to include or not include a class or method, providing decision model information using standard UML notation. The team built two applications, one for a low Earth orbit, and one orbiting a libration point at a greater distance from the Earth.

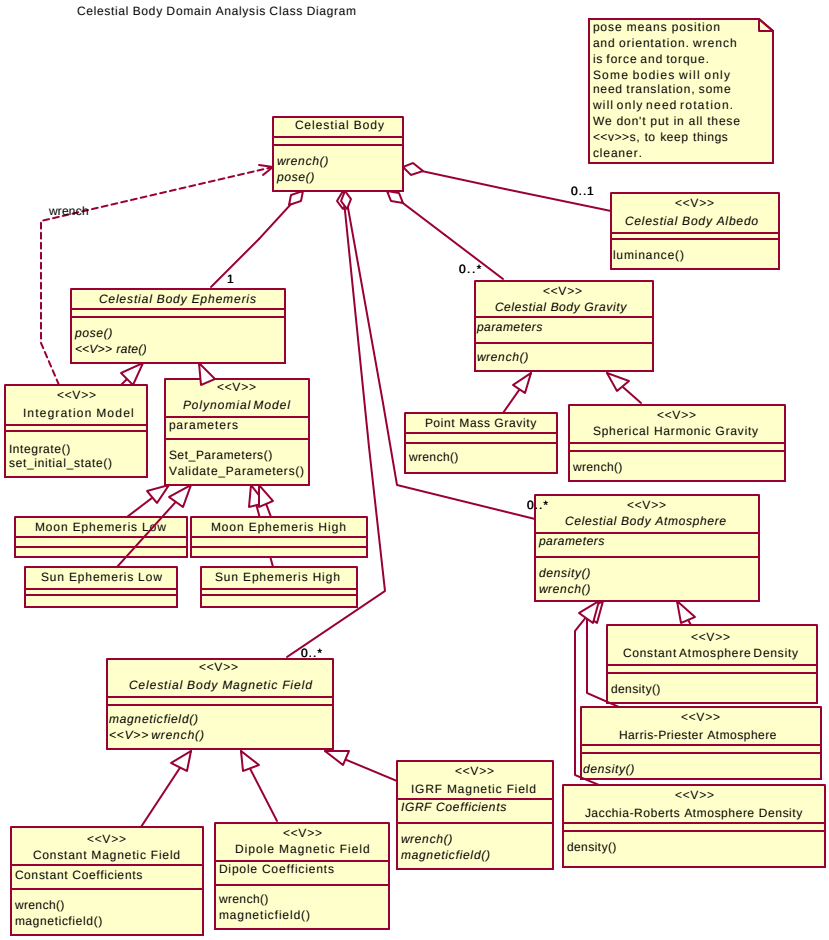

Figure 5. Celestial Body Subdomain Class Diagram

The differences in mission orbit mapped to different selections being made when applying the decision model. This was a simple and successful first test; further prototypes will evaluate whether this concept works with larger, more complex applications.

As the <<V>> stereotypes are used in the context of an object-oriented notation, the use of stereotypes interacts with concepts such as inheritance and aggregation. Rules for the interaction of variabilities and object-oriented concepts are described in [8]. These stereotypes and rules cover inclusion decisions, but they don't cover more complex constraints such as "only use objects of class Moon Ephemeris Low if it is contained by a Celestial Body object representing the Moon." We need to investigate whether UML notation is the best approach to representing such rules.

Figure 6 shows a mission/client matrix. Any model in the domain (in this example, a coarse sun sensor) may have such a matrix. The rows represent the legacy missions being analyzed for the product line, and the columns represent other models within the domain using the given capability. Each cell shows how a particular client model used the model within a particular mission's flight software. The mission/capability matrix is similar in concept, with each column representing a capability instead of a client model, and with the cells containing description of how a particular mission implements the capability.

Capability using CSS model

Mission	Estimation	Control	Mode/Constraint Management, Telemetry
WIRE	*Ac_USun_BF* • TRIAD • Kalman Filter Residuals	*Ac_USun_BF* • Digital Sun Control (Compute precession moment) • Momentum Control	*SunPresent(any eye lit)* • Eclipse logic
SWAS	*Ac_USun_BF* • TRIAD • Kalman Filter Residuals	*Ac_USun_BF* • Digital Sun Control • Momentum Control	*SunPresent(any eye lit)* • Eclipse logic

Figure 6. Mission/Client Matrix

These matrix representations were first tried on the second prototype application; for the first prototype we used diagramming techniques that worked for the orbit propagator, as there are few if any variations in how these models are implemented. However, the sensor models needed for the second prototype had more subtle differences in implementation details, which were difficult to capture in a diagrammatic notation. We found it more useful to use a simple table and supplement it with text descriptions as needed, rather than trying to use a notation that over-constrains the problem.

A second factor illustrated by these matrices is the need to focus on who will be reading various products. These tables are intended for domain engineers to compare and contrast past missions. The tabular notation strongly supports summarization of information from many missions, but relies on domain engineers having the knowledge needed to fill in the details from these summaries. An application engineer would not use these matrices for mission analysis and design; a separate notation specific to the task of applying the decision model will be used.

Observations

As a result of the prototyping, we can make both general observations about product line development and some specific conclusions about the first two prototypes. The first prototype has been completed and evaluated, and the second prototype is nearing completion. The second prototype also provides some lessons, but it still is awaiting a full evaluation. We can observe the following so far:

- As described in the last section, we were able to see a variability in a mission profile reflected in the selection of models from the Celestial Body subdomain.
- We did not make a clear distinction between domain and application engineering processes and products. However, the reusable components and the application specific components can be identified after the fact by examining the prototype 1 source code. The next step is to fully document the domain and application engineering product templates and processes; and then to apply them to subsequent prototypes.
- We observed variations in design approach between the first and second prototype application. One key difference was that the first prototype used direct message passing between objects, where the second prototype used "switchboard code" to read sensor data and then call the sensor data processing class, rather than having the data processing class call the read directly. This difference will add complexity to the application engineering process; so it is worth analyzing further to determine whether the difference is inherent in the problem domain, or simply an ad hoc variation.
- Despite our recognition that we need to involve all the different GNC stakeholders in the prototyping effort, we have focused so far on the developer perspective. This points to a need to revisit our prototype plan and explicitly address the issues associated with key stakeholders. The initial set to consider includes controls analysts who define GNC requirements for missions, and flight software testers.

This leads into some more general observations. One is that the functionality, architecture, and process issues are *all* important to product line development, and that they are all coupled. An example of this is the coupling between the architecture and the application engineering process. To make application engineering efficient, one should create reusable products that are easy to understand, and define a decision model that is as easy as possible to automate. Both of these goals are easier to accomplish if the architecture has a standard format for implementing modules (classes in the case of object-oriented development). This may not be fully attainable in the context of a given problem such as flight GNC software, but it should be a design goal. In short, most product line approaches allow a lot of flexibility in defining process and product details. The choices made for these processes and products should be driven by the organization's goals and then adhered to.

Another general observation is that one needs to repeat the domain assessment for each new product line release. This was one major difference between Synthesis, which tended to put planning activities outside the domain engineering lifecycle, and FAST or

PuLSE, which make the business analysis and planning an integral part of the analysis activities for a release. Product line engineering is a long-term investment that needs to be re-evaluated frequently to determine whether you are still creating the products with the highest payoff. In the NASA context, these assessments may change as the set of anticipated missions changes via winning proposals or having a mission cancelled. This is not a novel idea, but it is an important one that has not always been carried out.

Conclusions

The factors that we believe will make a product line effort succeed are as follows:

- Develop a series of increasingly realistic prototypes with the goal of ultimately demonstrating a small but realistic application in the target environment. In the case of the FSB this will require running on test versions of flight hardware.
- Keep the architecture and the decision model as simple and consistent as the application domain will allow. This will make the application engineering process easier to carry out and ultimately to automate.
- Consider the perspectives of and the tasks carried out by all users of the product line on both the domain engineering and application engineering side of the process.

The next step is to analyze the second prototype and to document product line processes and products based on the results of the first two prototypes. This work will include expanding the study to determine the needs of controls analysts and flight software testers and incorporating processes and products that are designed based on this analysis. This fully documented product line approach can then be applied to more realistic prototypes to determine whether it is feasible and beneficial to the FSB and GSFC Projects.

References

1. Software Productivity Consortium, *Reuse-Driven Software Processes Guidebook*, SPC-92019-CMC version 02.00.03 November 1993.
2. Weiss, David M. and Chi Tau Robert Lai *Software Product-Line Engineering: A Family-Based Software Development Approach*. Addison-Wesley, 1999.
3. Bayer, J., O. Flege, P. Knauber, et al., “PuLSE: A methodology to develop software product lines”, Symposium on Software Reusability (SSR99), May 1999.
4. Kang, K., et al. *Feature-Oriented Domain Analysis (FODA) Feasibility Study* (CMU/SEI-90-TR-21, ADA 235785). Pittsburgh, PA: SEI CMU, 1990.
5. McComas, D., J. O'Donnell, Jr., and S. Andrews, "Using Automatic Code Generation In the Attitude Control Flight Software Engineering Process", Proceedings of the 23rd Annual Software Engineering Workshop, December 1998.
6. Condon, S., R. Hendrick, M. E. Stark, W. Steger, "The Generalized Support Software (GSS) Domain Engineering Process: An Object-Oriented Implementation and Reuse Success at Goddard Space Flight Center", Addendum to the Proceedings of the Conference on Object-Oriented Programming Systems,

Languages, and Applications (OOPSLA 96), San Jose,California, U.S.A., October 1996

7. Rumbaugh, James, Ivar Jacobson, Grady Booch, *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1998.
8. Morisio, M., Travassos, G.H., Stark, M. “Extending UML to Support Domain Analysis”. IEEE International Conference on Automated Software Engineering – ASE’00. Grenoble, France, 2000.
9. McComas, D.; Leake, S.; Stark, M.; Morisio, M.; Travassos, G.H., White, M. “Addressing Variability in a Guidance, Navigation, and Control Flight Software Product Line”. Proceedings of the First Software Product Line Conference. The First Product Line Conference – SPLC1. Denver, Colorado, 2000